ADVANCES IN THE INTEGRATION OF LARGE DATA SETS FOR SEISMIC MONITORING OF NUCLEAR EXPLOSIONS

Dorthe B. Carr[1], Jennifer E. Lewis[1], Sandy Ballard[1], Elaine M. Martinez[1], Jeff W. Hampton[1], Bion J. Merchant[1],
Richard J. Stead[2], Michelle Crown[3], Jorge Roman-Nieves[3], and John J. Dwyer[3]

Sandia National Laboratories[1], Los Alamos National Laboratory[2], and Air Force Technical Applications Center[3]

Sponsored by National Nuclear Security Administration
Office of Nonproliferation Research and Development
Office of Defense Nuclear Nonproliferation

Contract Nos. DE-AC04-94AL8500[1] and DE-AC52-06NA-25396[2]

ABSTRACT

The National Nuclear Security Administration (NNSA) Ground-Based Nuclear Explosion Monitoring Research and Engineering (GNEMRE) program has been integrating large sets of seismic events and their associated measurements for almost a decade to support nuclear explosion monitoring. During that time the integration process has changed significantly, generally becoming more complex and more automated as the number of events and the range of associated measurements has steadily grown. In this paper, we explain the methodology for integrating database tables from different products that are part of a Knowledge Base (KB) release.

The major effort of KB integration is merging events and their associated information in Oracle database tables. We have developed a substantial foundation of structure and software to assure data integrity in the integration of diverse data sets. The structural part of this foundation utilizes Oracle data dictionary tables along with complementary custom database tables. These custom tables contain information specifically related to how KB database objects are built. Information such as descriptions and database types are stored in these custom tables so that the KB structure is easily modifiable making it more flexible than it would be following a traditional database design. This “metadata” of the supporting structures is called the “schema schema,” and all the integration tools are based on this structure.

Los Alamos National Laboratory (LANL) has developed a tool to make sure the information in the product database tables is accurate and valid. Called the Quality Control Tool (QCTool), this tool checks that the database tables conform to the database structures found in the “schema schema” and that the values in the database tables are reasonable. Database tables are not merged together until the errors generated by QCTool are either corrected or explained.

Sandia National Laboratories (SNL) has developed a suite of merging tools that are part of a set of generalized database interface tools called DBTools. These tools are based on the concept of a “RowGraph,” which builds a set of information for an event based on the specified relationships. Using RowGraphs the applications in DBTools can then easily read and write a complete set of information. There are four major DBTools that are used to integrate the database tables for the KB: EvLoader (Event Loader) merges events and their associated information from different KB products into one complete set of database tables without duplicating information and updating IDs so that there are no conflicts, WFMerge (Waveform Merge) merges flat file waveforms and their associated database information using correlation parameters to determine if waveforms are the same, DTX (DaTa eXchange) is used to merge the station parameter information, and Remap is used to create tables that remap IDs from one set of tables to another set of tables. For example, Remap is used to remap IDs from one version of the KB to another version and from the KB to other database accounts. Two other tools in DBTools that can be used in manipulating database tables in the KB are DBCompare and Unloader. DBCompare is used to compare tables for equality as defined by the tool user. Unloader deletes one or more specified rows from the database and also deletes all rows that are connected to the specified row(s) according to specified relationships. Rows are only deleted if doing so will not violate any foreign key relationships in the database, i.e., no rows are left orphaned. Using the QCTool and DBTools to check and merge information we are assured of a complete quality set of referenced database tables in the KB.

Report Documentation Page	*Form Approved* *OMB No. 0704-0188*

1. REPORT DATE **SEP 2007**	2. REPORT TYPE	3. DATES COVERED **00-00-2007 to 00-00-2007**

4. TITLE AND SUBTITLE **Advances in the Integration of Large Data Sets for Seismic Monitoring of Nuclear Explosions**	5a. CONTRACT NUMBER 5b. GRANT NUMBER 5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER 5e. TASK NUMBER 5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Los Alamos National Laboratory,PO Box 1663,Los Alamos,NM,87545**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR’S ACRONYM(S) 11. SPONSOR/MONITOR’S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES

Proceedings of the 29th Monitoring Research Review: Ground-Based Nuclear Explosion Monitoring Technologies, 25-27 Sep 2007, Denver, CO sponsored by the National Nuclear Security Administration (NNSA) and the Air Force Research Laboratory (AFRL)

14. ABSTRACT

see report

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**	**Same as Report (SAR)**	**8**	

OBJECTIVES

One process for integrating a Knowledge Base (KB) is to take Oracle database tables with seismic event information and their associated measurements from different product integrators and merge the information into one set of complete tables. With each release of the KB, the process has become more complex as the number of tables and the size of the tables have both increased. We have developed a process to merge all the different database information into a complete set of quality referential database tables in the KB, using a number of tools designed at SNL and LANL.

RESEARCH ACCOMPLISHED

The NNSA Knowledge Base consists of Oracle database tables, flat-file information, and tools. The bulk of the KB is in the database tables. The core database tables hold seismic event information – origins, arrivals, associations, and magnitudes that come from various global, regional, and local bulletins. Other database tables include custom parameter information for location, event identification, and coda magnitudes that is generated from research. There are database tables that contain waveform metadata that point to flat-file waveforms on the system. Finally, there are database tables that contain information about the seismic stations – locations, channel names, instrument responses, etc. The major integration task for the KB is to combine database tables from different sources (product integrators) into one set of complete tables.

SNL needed to find a good way to merge the Oracle tables from the different sources without creating duplicate records. There needed to be proper links maintained between the related information, and SNL needed to make sure the information in the tables was accurate and correct. What evolved was a set of database integration tools. Richard Stead at LANL (Stead et al., 2006; Stead, 2007) developed the QCTool used for quality assurance and quality control. Sandy Ballard and Jennifer Lewis at SNL (Ballard and Lewis, 2004; Ballard and Lewis, 2007) developed DBTools to manipulate the information in the database. The six DBTools applications are used to compare and merge information in different database tables: EvLoader, WFMerge, DTX, Remap, DBCompare, and Unloader.

Schema Schema

Both the QCTool and DBTools depend on knowing the particulars of the database tables that are being QC'd and manipulated. This is accomplished with a database schema that holds table definition information for the KB schemas or what has been termed: “schema schema.” The schema schema concept was deliberately designed by Richard Stead to have a close relationship to the Oracle data dictionary, to make the use and understanding of the schema schema simple and straightforward. However, it does go well beyond the Oracle data dictionary in various ways—in particular, because it exists apart from the tables it describes and it exists at all times regardless of which objects are currently defined or how they are defined. Any schemas can be described in the schema schema tables.

There are four tables in the “schema schema.” The table **tabdescript** provides descriptions of the tables defined in each schema. Table descriptions are meant only to provide a text description of the table and its affiliation to a schema. The **coldescript** table provides descriptions of the columns defined in each schema. Column descriptions are meant to provide all data necessary to fully document the column and provide for quality control of the column. Only one column definition per column name per schema is permitted. The table **colassoc** links the columns described in the **coldescript** table to the tables described in the **tabdescript** table in such a way as to provide column ordering, roles, and information on keys within the context of a particular table. The **colassoc** table also says whether the table described is an “idowner” table, meaning that the primary key is an ID and the unique key defines the ID. The **glossary** table defines all abbreviations, acronyms, and other odd names used throughout the descriptions and also define the legal entries for columns with range types of defined and finite set. For the KB, there are two major defined schemas: NNSA KB Core and NNSA KB Custom. However, the CSS3.0, USNDC P2B2, and USNDC P3 schemas are also captured in the schema schema tables. The descriptions and definitions of the schema schema tables and columns can be found in the NNSA KB Database Guide (Carr, 2006).

QCTool

QCTool (Quality Control Tool) is an automated quality assurance and quality control tool for database tables. The tool executes three kinds of QC checks: single-table, multi-table, and complex joins. The single-table check verifies that the table matches the documented table structure described in the schema schema tables and provides the QCTool access to a complete description of that table. The defined primary key for the table is validated, along with any unique keys that exist for that table. Then each column in the table is checked. The maximum and minimum values are found. If the values in the columns are strings, they are checked to make sure they match defined terms. If NA values are allowed, the number of rows that have the NA value is determined. If the values in the column are numeric, they are checked to make sure they fall within the defined range. None of the checks may yield errors, but they can indicate errors when subsequently reviewed.

The three kinds of multi-table checks are single-column cross reference between tables, two-column cross reference between tables, and indirect cross references. A single-column cross reference requires that every value in the specified column of the table being checked is also found in the specified column of the table that the first table is to be compared against. The two-column check is the same as the single-column check, except that each unique pair of values from the two specified columns in the table being checked must be found in the two columns specified for the comparison table. The indirect cross-reference check handles cases where the reference for a column in the table being checked is specified by a second column in the same table. The archetypical example of this is the **wftag** table, where the reference for the *tagid* column is specified by the *tagname* column; that is, if *tagname* equals “evid,” then *tagid* contains an *evid* and should be compared to columns in other tables that are *evids* (like **event**.*evid*), but if *tagname* equals “arid,” then the contents of *tagid* must be compared to *arids* (like **arrival**.*arid*).

The third and final checks are the complex joins. These are specified in an auxiliary table called **complexjoin**. The **complexjoin** table supports QC operations by providing a mechanism to specify any consistency requirements or expectations across as many as three generalized tables. There is always a target column for validation, and additional tables can be defined. The operation applied to the target column that is required to be true is specified in the *joinop* column, and it is a standard SQL operator. The *clause* column provides the portion of the where clause that follows *joinop* and will often include subqueries and the special strings ‘&t1,’ ‘&t2,’ and ‘&t3’ for the three tables. Some simple examples for the KB include verifying that a date in a table corresponds to the time specified in the same table, verifying that the contents of a field in one table correspond to those in another when there is no direct link between the tables (such as **wfdisc**.*instype* versus **instrument**.*instype*), and verifying that a count in a field of one table equals the count of those objects in another table (such as **origin**.*ndef* versus the count of time-defining arrivals in **assoc**).

DBTools

DBTools is a suite of applications for manipulating information in a relational database in an intelligent way. This means that DBTools (1) does not duplicate information already in the database, (2) links new information to related information already in the database, and (3) remaps identification numbers on the input data to prevent conflict with identification numbers already in the database. All the applications depend on a library of utilities called DBUtilLib.

DBUtilLib is based on the concept of a RowGraph. A RowGraph manages a set of connected rows from a database. Mathematically, this set of rows forms a directed graph consisting of a set of vertexes (the rows) and edges (one-way connections between pairs of vertexes). Starting from any row, it is possible to access every other row that is connected to the starting row. The relationships that define how the rows are related are specified by the user. Once a set of database tables has been identified and the relationships between those tables defined, one may specify some initial row in the database and extract all other rows in all the other tables in the database that are related to the initial row by one or more defined relationships. DBUtilLib is written in a completely generic manner. No information is hard-coded about any particular set of tables. Instead DBUtilLib uses the schema schema as the way to determine the objects and their structure at run time. This approach makes the tools that use DBUtilLib immune to most schema changes such as addition/deletion of columns from a table or changes in the size of a particular data element, such as going from i8 to i9.

DBTools has six major applications: EvLoader, WFMerge, DTX, Remap, DBCompare, and Unloader.

EvLoader (Event Loader) merges one or more rows from a source event table into a target event table. All information in the database that is linked to the source event row(s) (such as origins, arrivals, etc.) is also merged based on how the user specifies what information is related to the source event in the parameter file. The code operates in two modes. In the first, origins in the source event are merged with origins in the target event based on *evid* number. In the second mode of operations, source events are merged with target events based on spatial/temporal correlation. Origins that are members of the same event in the source tables will remain members of the same event in the target tables.

WFMerge (WaveForm Merge) is a waveform merging application. This application handles the merging of binary waveform files and their associated **wfdisc** database table information. The **wfdisc** table contains metadata about the waveforms, while the waveforms themselves are stored on disc. WFMerge can also handle **wftag** table information associated with the **wfdisc** table by keeping **wftag** rows in synch with their corresponding **wfdisc** rows and by using a remap table to remap the tagids in the **wftag** table.

DTX (DaTa eXchange) merges data from one data source into another data source. DTX recognizes three information storage formats: a set of database tables, a set of ASCII flat files, and XML file that has many tables and their associated data within one file. These three formats correspond directly to the three different data access types recognized by DBUtilLib. DTX can convert information from any one of these formats into any of the others. Both formats can be the same, as might be the case if DTX is moving data from one set of database tables to another set of database tables.

Given two sets of database tables, the Remap application will generate a remap table that will relate the identification numbers in the source tables to the identification numbers in the target tables of the same type. When researchers at the NNSA laboratories send database information to other labs, the identification numbers are generally renumbered when the information is merged into the other laboratory’s database. This utility will allow the researchers to discover the identification numbers in the other lab’s database that equate to their original identification numbers.

Unloader deletes one or more specific rows from the database and also deletes all rows that are connected to the specified rows(s) according to relationships either specified in the parameter file or deduced from the table definition table information (schema schema). Rows are only deleted if doing so will not violate any foreign key relationships in the database, i.e., nothing else is still pointing to them.

DBCompare is an application for comparing tables for equality. The user defines the equality. There are four types of table comparisons that DBCompare can handle: (1) idowner tables, (2) non-idowner tables, (3) remap tables, and (4) all tables. DBCompare compares the data in all of the tables in the source schema with tables of the same type in the target schema. It then does the reverse of that and compares the data in all of the tables in the target schema with tables of the same type in the source schema. It is also possible to compare only the source to the target, but not the target to the source. If all of these comparisons succeed, then all of the tables in both schemas are the same.

KB Integration Process

The major part of KB Integration is to merge database tables from individual product integrators into one set of tables. The product integrators send their database tables to the KB Integrator. The tables are checked to make sure they comply with established standards by using QCTool. Then the tables are merged into the target tables using EvLoader, WFMerge, DTX, and Remap (Figure 1). The following text goes into details about the process.

Check Data with the QCTool

The first step in the process is to check the data in the database tables with the QCTool. There is a master KB QCTool parameter file that has all the multi-table checks that are necessary to determine that the tables in the KB have clean consistent data. That parameter file is copied and edited to work with the specific tables in the dataset. Because a standard parameter file is used, the editing is minor. Some of the potential changes are to define the prefixes and suffixes for the table names, to add the correct username and password to access the database, and to define the output file names. The **complexjoin** table is also standardized, so unless the product integrator sends new rows for the table, it is considered to be correct. The QCTool does check against the **glossary** table when it does the single-table checks. Most errors found when checking against the **glossary** table are missing definitions. The column *auth* is usually the problem. The product integrator sends the **glossary** table as part of the delivery, and any new information must be inserted into the **glossary** table used by the KB Integrator before running the QCTool. The final step is to set all the primary and unique keys on the tables that will be checked.

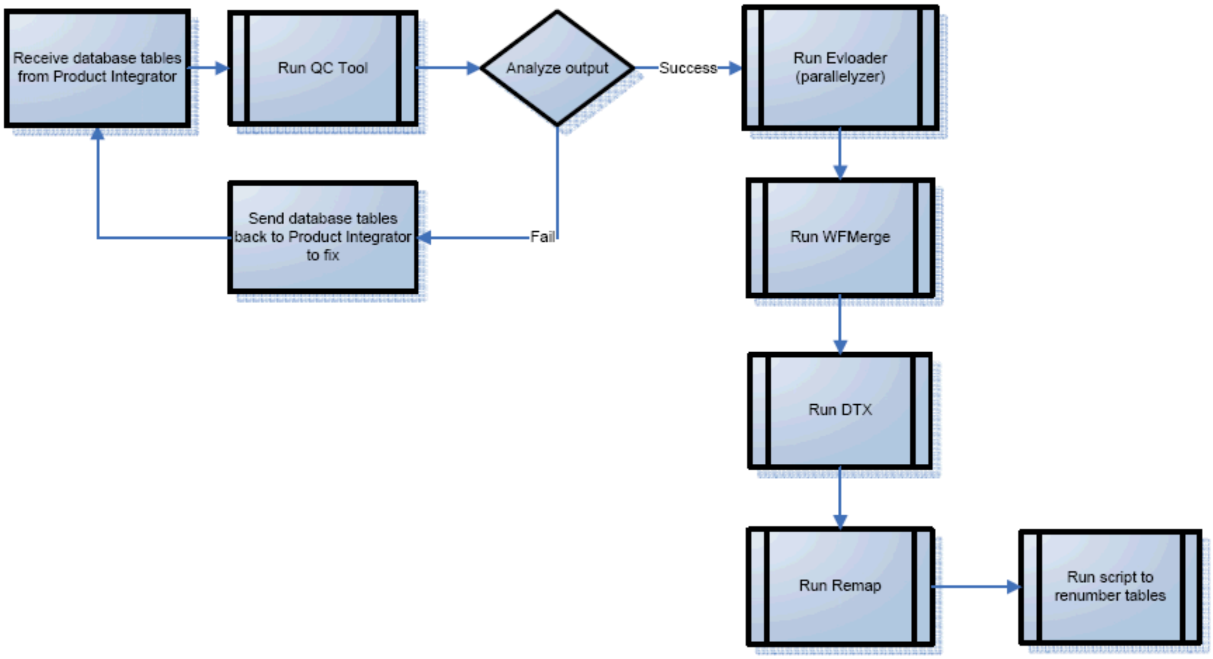
Receive database tables from Product Integrator
Run QC Tool
Analyze output
Success
Run Evloader (parallelyzer)
Fail
Send database tables back to Product Integrator to fix
Run WFMerge
Run DTX
Run Remap
Run script to renumber tables

Figure 1. KB integration process.

The QCTool is run using a simple command – "qctool parfilename.par." The first check that is done is to make sure that the tables follow the NNSA KB Core or NNSA KB Custom schemas as defined in the schema schema tables. If they do not, QCTool immediately exits without doing any other checks. If the tables match the schemas, the QCTool continues by first doing the single-table checks. It writes the information from these checks into a file "single-table.out." The structure of the output file is very easy to navigate. The tables are checked in alphabetical order. First the primary and unique keys for the table are confirmed. If they cannot be set, this is the first error message for that table in the "single-table.out" file. Then each column in the table is listed. The NA value for the column is printed, and there is a statement about whether the column is allowed to have the NA value. Then the max and min values are listed to give the user a range, and the top two most common repeated values are listed. If the QCTool finds a problem, such as a value that does not match the range defined in the schema schema tables, that error is listed out, along with the SQL query that was used to discover the error.

The second checks that are done are the multi-table checks that are documented in the parameter file. The output from these checks is in the file "reference.out." Only errors are written to the output file. Again, the query that was used when the errors were found is provided. Any of the checks that are skipped, usually because one of the tables being checked does not exist, are noted. The complexjoin checks are last, and they are recorded in the output file "complexjoin.out." This file lists all the joins that are run from the **complexjoin** table. If there is an error, the error is noted along with the query used to generate the error.

The KB Integrator looks at the output files from the QCTool run and notes any errors. If the QCTool was run at the product integrator's site (and the KB Integrator encourages this), then the KB Integrator compares the errors found running QC Tool to errors noted by the product integrator. Many errors, especially those concerning the range of values in a column, can be noted and listed as caveats. If the tables have errors not described or explained by the product integrator, the tables go back to the product integrator to fix. It is only when the QCTool has been run successfully, i.e., all errors noted by the QCTool are either fixed or explained and documented, that the information in the tables can be merged into the KB target tables.

Merge catalog and parameter data

The bulk of the database information is seismic event information (origin, arrivals, magnitudes) that comes from various earthquake catalogs, and parameter data developed by the product integrator to help detect, locate, and identify events. This information is merged using EvLoader.

As with QCTool, there is a master KB parameter file for EvLoader. The most important part of the parameter file for EvLoader, and any of the other DBTools, is the defined relationships between the tables in the schema. A relationship is how tables relate to one another within the schema. A relationship can exist between two, and only two, tables. If a table is related to more than one other table, it must have one relationship defined for each table it is related to. To define a relationship, a *relationship id*, *source table*, *target table*, *relationship where clause*, and *constraint identifier* are required. The *relationship id* is a string that identifies the relationship, and the applications are not concerned with its actual value; it is mostly used to display useful information about the relationship. The *source table* is where the relationship between two tables originates. The *target table* is where the relationship between two tables ends. The *relationship where clause* is a SQL where clause that defines what columns in the two tables involved in the relationship must be equal for two rows from the tables to be related. The *constraint identifier* identifies the number of rows required and/or allowed to be returned as the result of executing a SQL SELECT statement that uses the *relationship where clause* on the *target table* for the relationship to be valid. All the relationships that can be defined between tables in the NNSA KB Core and NNSA KB Custom schemas that hold seismic event information and their associated parameters are in the master KB parameter file for EvLoader.

The master KB parameter file is copied and edited to work with the tables in the data set. Again, because a standard parameter file is used, the editing is minor. The source tables, target tables, and origin ranking table need to be defined. The origin ranking table is used by EvLoader to determine the preferred origin in the **event** table. When merging events into the KB, we use the spatial/temporal correlation mode of EvLoader. The way EvLoader works is that target events are selected as potential recipients of source origins if their preferred origin falls within defined spatial and temporal ranges. The spatial and temporal ranges are defined by setting a maximum correlation distance and a maximum correlation time. The maximum correlation distance and maximum correlation time can be different for different catalogs. EvLoader is set up so that there can be one set of spatial and temporal defined ranges for global catalogs and another for regional catalogs. These values are defined in the parameter file, along with a list of the catalog authors considered to be regional catalogs.

Once the parameter file is finalized, constraints and indexes are set on all the source and target tables. EvLoader will run faster if the correct indexes are on the table. Then sequences are created for all the IDs in the tables that will be remapped. As IDs need to be remapped in the merge process, EvLoader will use the next number in the sequence for that particular ID. The number at which the sequence starts is determined by looking at the maximum values of the IDs in the target tables.

EvLoader can be run in two ways. The first runs the program sequentially through the events in the source table. The command to use is – “evloader parfilename.par.” Each event is merged sequentially, and it takes anywhere from 100ths of a second to a few seconds to merge an event, depending on how many arrivals and amplitudes are attached to the event. If you have small tables, this is a reasonable way to use EvLoader. However, if you have large tables, it can take many hours for EvLoader to finish. The way to speed up the time it takes for EvLoader to merge large database tables is to run it in parallel. The command to use is – “parallelyzeevloader parfilename.par #”, where # is the number of processes to initiate. When using the parallelyze option, a temporary table is created that counts the number of arrivals and amplitudes for each event. The total number of events is then divided into groups that have a similar number of arrivals/amplitudes. The number of groups is the number of processes that are to be initiated. The program copies the parfilename.par file so that there is one parameter file for each process. Each parameter file will merge one specific group of events. EvLoader is then started using each parameter file simultaneously.

For example: The source data set has 10,000 events with a total of 350,000 arrivals and 100,000 amplitudes. The parfile evloaderA.par is created to merge the data set with EvLoader. If the events are merged sequentially, it can take many hours, if not days, to merge the entire data set. To merge the events in parallel, use the command “parallelyzeevloader evloaderA.par 10.” A temporary table is created that counts up the arrivals/amplitudes for each event and then breaks the events into 10 groups that each have approximately 45,000 arrivals and amplitudes [(350,000 + 100,00)/10]. Ten parameter files are created from evloaderA.par, one for each group, and then are started simultaneously. Because there are 10 roughly equal groups, the merge should take place in a much shorter time period, and all the groups should finish around the same time.

EvLoader merges one event at a time. Using the distance and time correlation parameters defined in the parameter file, EvLoader either inserts the event and its associated information as a new event in the target table or adds it to an existing event in the target table. Once the merge using EvLoader is completed, there are a number of tests to determine how well the merge went. The first is to compare the number of rows in the source tables to the number of new rows in the target tables. It is expected that, in most cases, all the rows from most of the source tables will be added to the target tables. The number of rows in the event table is dependent on how EvLoader decided to merge the source events into the target events, so it may not be a one-to-one match. Also, if events/origins in the source tables being merged into the target tables are already in the target tables, they will not be merged again. If there are discrepancies seen when looking at the number of rows in the tables, then DBCompare and Remap are used to compare the information in the target tables to the information in the source tables. If it is determined that events were merged incorrectly, then Unloader can be used to remove information from the target tables. If the EvLoader merge is successful, then the next step is to merge the waveforms and their associated metadata.

Merge waveforms

Once the catalog and event parameter tables are merged, the next step is to merge the waveforms and their metadata found in the **wfdisc** and **wftag** tables using WFMerge. There is a master KB parameter file for WFMerge that defines the relationships between the **wfdisc** and **wftag** tables. This file is copied and edited to work with the **wfdisc** and **wftag** tables and waveforms in the data set. As with EvLoader, edits include defining the source and target tables. Three additional tables, **source_remap** table, **idgaps** table, and **backup_wfdisc** table are also defined. The **source_remap** table is used to remap *tagids* in the **wftag** table. The **idgaps** table is used by WFMerge to get new IDs when the source tables are merged into the target tables. When a row in the target **wfdisc** table is modified or deleted, it will be saved to the **backup_wfdisc** table. If no **backup_wfdisc** table is specified, no backups will be made. WFMerge merges database information, but also merges waveform files. The copy of the parameter file is edited to define the source and target waveform directories and the waveform name format. Correlation, timeshift, samprate tolerance, and calper tolerance parameters used for merging the actual waveforms on disk are specified. Finally, the **source_remap** table is created by using information from the EvLoader merge.

WFMerge is run using the command – “wfmerge parfilename.par.” Each waveform is merged one at a time. If there is no overlap between the source **wfdisc** row/waveform and the target **wfdisc** row/waveform, then the *wfid* in the source **wfdisc** row is updated and the row is inserted into the target **wfdisc** table. The source waveform is copied to the target waveform directory. If the source **wfdisc** row/waveform overlaps with some target **wfdisc** row/waveform, then first, the information in the source **wfdisc** row and target **wfdisc** row are merged into a single **wfdisc** row and written to the **target** wfdisc table. The old wfdisc rows from both the source and target are written to the **backup_wfdisc** table. Then, several checks are performed on the source and target waveforms to make sure they are consistent with one another before the waveforms are merged. If any of the checks are violated the waveforms will not be merged.

After WFMerge is completed, checks are run to make sure the merge was successful. Again, the first test is to compare the number of new rows in the target table to the number of rows in the source table. There will not be a one-to-one match if any of the source waveforms overlapped with target waveforms, but the comparison does provide useful information. If there appears to be a problem when comparing the number of rows in the tables, then DBCompare and/or Remap can be used to figure out what happened during the merge. If WFMerge successfully merged the waveforms and their metadata, the next step is to merge the station parameter information.

Merge station information

DTX is used to merge the station parameter information. The major problem in merging these tables is that only two of the six station parameter tables defined in the NNSA KB Core schema are “idowner” tables. What is meant by this is that there are no numeric ID primary keys for the **affiliation**, **network**, **sensor,** and **site** tables. Therefore, the first step in the station parameter merge is to create custom source and target station parameter tables that look like the original tables, but contain numeric IDs that are primary keys and foreign keys. The script *create_idowner_station_tables.sql* is used to do this. The custom station parameter tables created by the script are the tables used to merge the station parameter information.

The master KB parameter file for DTX contains the defined relationships between the custom station parameter tables. This parameter file is copied and edited to work with the tables in the data set. The only edits that are really needed are to define the source and target tables. The command – “dtx parfilename.par” is used to start DTX. The information is merged by creating RowGraphs containing the source data and adding new information to the target tables that does not duplicate information in the target tables. Once the merge is completed, the first test is to compare the number of rows added to the target tables to the number of rows in the source tables. It is expected that the numbers should be very close, if not the same. If the numbers are not the same, then DBCompare and Remap can be used to compare the information in the target tables to the information in the source tables. Once the DTX merge is considered to be successful, the station parameter target tables are changed back to the structures defined in the NNSA KB Core schema. The script *change_original_station_tables.sql* is used to change the tables back. At this point, all the database tables have been successfully merged.

Remap IDs

The final step in the integration process is to remap the IDs in the new KB target tables to the IDs in the prior release of the KB. This is done so that the user of the KB doesn’t have to deal with changing IDs every time a KB release is made. The program Remap compares a source table to a target table and generates a remap table of IDs based on a user defined equality. For example, the *orids* in a source and target **origin** table are remapped based on *lat, lon, depth, time,* and *auth*. As stated earlier, idowner tables are tables where an ID is the primary key. There are master KB Remap parameter files for remapping each of the idowner tables in the KB database. Using the command – “remap parfilename.par,” with each idowner master remap parameter file, each idowner source table (prior KB release) is remapped to the idowner target table (new KB release). Scripts are then used to change the IDs in the new KB release tables to the IDs for that information in the prior release.

Once the IDs have been remapped, the Oracle database tables are ready to be tested with the other information in the KB to make sure that the integration process was successful and the products in the KB work as expected.

CONCLUSIONS AND RECOMMENDATIONS

We have developed a process to integrate large Oracle database tables that contain seismic data and produce clean, consistent, and useful information. To do this process effectively, we have developed a suite of tools that make it easy to QC and manipulate the data in the database tables. We also have tools that can be used to check that the merge process was successful. The tools are written in a generic manner and use the schema schema tables to define the schema of the database tables that are being accessed by the tools. So while we are using the tools to QC and merge seismic data, they can be used for any kind of data that has a schema defined in the schema schema tables.

ACKNOWLEDGEMENTS

The authors wish to acknowledge Chris Young and Mark Harris for their encouragement on documenting this process.

REFERENCES

Ballard, S., and J. Lewis (2004). DBTools: A Suite of Tools for Manipulating Information in a Relational Database, in *Proceedings of the 26th Seismic Research Review: Trends in Nuclear Explosion Monitoring*, LA-UR-04-5801, Vol. 2, pp 700–709.

Ballard, S. and J. Lewis (2007). DBTools Help Document, Sandia National Laboratories, in preparation

Carr, D. (2006). National Nuclear Security Administration Knowledge Base Database Guide (2006). Sandia National Laboratories, SAND2004-0961P.

Stead, R., M. Begnaud, and J. Aquilar-Chang (2006). Advances in Data Integration and Quality Control in Support of the NNSA Knowledge Base, in *Proceedings of the 28th Seismic Research Review: Ground-Based Nuclear Explosion Monitoring Technologies,* LA-UR-06-5471, Vol.2, p. 1028–1037.

Stead, R. (2007). QCTool User’s Manual, Los Alamos National Laboratory, in preparation.